Assumptions, Domains, Types in Computer Algebra Systems

Richard J. Fateman
Nicholas Mitchell
University of California, Berkeley

July 10, 1992

Abstract

Computer algebra systems have limited success in implementing transformations requiring knowledge about symbols in formulas. Some of this is based on confusions between declarative and procedural specification.

It is easiest to explain this by means of a few examples.

1 Examples on Input

1.1 Sin($n\pi$)

Consider the rule

$$\sin(n\pi) \longrightarrow 0 \quad \text{if } n \text{ is an integer}$$

The meaning of this rule is rather clear to most readers of this paper. An argumentative person (or a literal-minded computer program!) might say: n is not an integer, but a non-numeric symbol, so the rule never applies. We must really be referring to the (infinite) set of rules that can be described by the *pattern* above. Thinking about the pattern declaratively, it presumably says that for every integer n assert the rule $\sin(n\pi) \rightarrow 0$. Rule-based computer algebra systems might need the additional rules that $\sin(0) \rightarrow 0$ and $sin(\pi) \rightarrow 0$ as well, since neither of those rules syntactically matches the form.

In particular, one of those rules can be directly applied to the expression $\sin(3\pi)$ to yield 0.

Do we wish to apply it to the expression $\sin(k\pi)$ where k is declared to be an integer, (even if k is not known to be a particular integer?) How about to $\sin(k \cdot (k+1)/2 \cdot \pi)$ where some computation may be required to realize that $k \cdot (k+1)/2$ is, indeed, an integer whenever k is an integer.

Or do we wish to apply it to the expression $\sin(x\pi)$ resulting in the "simplification" to z where

$$z = \begin{cases} 0 & \text{if } x \text{ is an integer} \\ 0 & \text{if } x \text{ is declared to be an integer} \\ \sin(x\pi) & \text{otherwise} \end{cases}$$

At first glance this seems unlikely, although one could imagine a sequence of operations that would tweezes out from a complicated expression those conditions which could be used to simplify it. We will see this situation in the section on Ouput.

1.2 Limits

The limit of $\exp(ax)$ as $x \to \infty$ is 0 if $a \leq 0$ (which in passing asserts that a is real), but is also true if a is complex, but $\mathrm{Re}(a)$ is negative. What should a system answer in the absence of information on a?

The puzzle is deeped if the question is the limit of $\exp(-ax)$ since the questioner (if human, anyway) has apparently expressed an intention that a is positive by placing a negative sign in front of it!

Computer algebra systems can provide a specific interpretation, but it must be a useful interpretation.

2 Examples on Output

We have already encountered in passing a situation in which the result of a computation has a side condition (that is, the example of $z = \sin(xpi)$ above). How general is this?

2.1 Integration

Should the result of $\int x\ dx$ be $x^2/2$? Or should it be an encoding of

$$x^2/2 + C$$

where C is a constant with respect to x? The reason we do not see computer integration routines producing these constants is that computer algebra systems are so clumsy in dealing with such data. The constants are almost universally discarded, even though the "constant" is sometimes quite important. In multiple integral the constant is in general an arbitrary function of all the other remaining variables of integration!

2.2 Solve

Consider the problem of solving $y = \sin(x)$ for x. Certainly one way of expressing the solution is $x = \arcsin(y)$, but this is not really adequate for some uses. For example, if a program can solve an equation then it should also be able to answer how many distinct solutions there are, and the details of each solution. Allowing for the convention that there is a principal arcsin denoted Arcsin, then $\arcsin(x)$ is the set of values $\{\text{Arcsin}(x) + n\pi \quad : \quad \text{integer}(n)\}$

Solving the expression $y = x\sin(x)$ then is similar to the previous problem except there is a "double zero" at $x = 0$. Should it be counted twice? What is the multiplicity of the zero of $y = x^n \sin(x)$ which might also have to make sense for negative n.

Having produced `Values` objects What more can be done with them?

The traditional algebraic manipulation programming approach would be to rewrite all the simplification and operational procedures that add, subtract, multiply, divide, integrate, (etc.) to account for this new data type. That is `Values[x : x>0] + y` should be transformed to `Values[x+y : x>0]`. It may be necessary to provide substantially more intelligence in combining conditions that would be implied by the naive example above.

The object-oriented approach would attempt to segment the changes to the program in a more data-oriented approach. That is, the basic programs would appeal to special handling routines whenever they encountered `Values` objects. In point of fact, this might not decrease the total amount of programming at all, but might provide better organizing principles for this and additional data types.

The axiomatic or algebraic approach would attempt to classify such new objects as elements of a formal algebraic structure such as a ring or a field. Some `Values` objects describe ranges or geometric figures, and can therefore participate in various well-known formal systems (group transformations, for example).

Here is a situation that appears to be only superficially more complicated than our example above, dealing with 2 range variables:

```
v1=Values[z : a<=z && z<=b]
v2=Values[z : c<=z && z<=d]
```

A correct set of answers for some simple manipulations might be:

```
v1+v2 -->Values[z : a+c<=z , z<=b+d]
v1*v2 -->Values[z : Min[a*c,a*d,b*c,b*d]<=x &&  z<=Max[a*c,a*d,b*c,b*d]]
```

Further computation with `v3=v1*v2` might lead to rather complicated expressions!

3 Type checking

Is there some classification that makes sense for type checking here? e.g.

```
f[x_Integer,y_Odd, z_Real,w_...]:=
```

that should be used to allow `f[k*(k+1)/2,...]` to pass?

Distinguish between types and domains. For example, 3.0 might be represented as a "floating point number" datum, but it is also numerically equal to the integer 3. The distinction between value and type should be made clearer. In Mathematica, 2 not rational because it happens also to be an integer. This seems wrong.

4 Further discussion

5 Conventions

Note that in mathematical discourse it is natural to use symbols to denote members of domains of undetermined value – n is an arbitrary natural number, z is sometimes used for an arbitrary complex-valued number. These are conventions, but usually not adequate for disambiguation of meaning. In textbooks one sometimes sees an initial "type binding" of symbols by typographic conventions. Such a declaration makes it apparent that $\mathbf{A}$ denotes a matrix, $\mathbf{a}$ is a vector, α is a scalar etc.

6 Binding of names

Who binds assumptions?

In Mathematica, the following construction is certainly unclear..

```
integrate[f_,{x_,a_,b_}]:=Block[{var=x},
                        assume [min[a,b]<=var<=max[a,b]]; ...]
```

What is the assumption talking about? The locally bound name var, the globar name var, the match-bound parameter x?

Can we clean this up with the Lisp model? (at least Lisp does not have to deal with some mysterious binding mechanism of f, x, a, and b, on some pattern association list).

The possibilities:

1. the global symbol has assumptions.
2. the local binding has assumptions.
3. assumptions are separately packaged in Contexts that can be inherited.

Additional hazards .. the meaning of `D[If[Integer[n],f[x],g[x]],x]` etc.

7 Proposed solutions

This section is conspicuously empty.

References

[1] Genesereth. Encyclopedia of AI?

[2] Wolfram. Mathematica

peoplesparc.Berkeley.EDU:fateman

stdin

Fri Jul 10 17:34:41 1992

cx / cx lw507 Evans

```
cx peoplesparc.Berkeley.EDU:fateman  Job: stdin  Date: Fri Jul 10 17:34:41 1992

cx peoplesparc.Berkeley.EDU:fateman  Job: stdin  Date: Fri Jul 10 17:34:41 1992

cx peoplesparc.Berkeley.EDU:fateman  Job: stdin  Date: Fri Jul 10 17:34:41 1992

cx peoplesparc.Berkeley.EDU:fateman  Job: stdin  Date: Fri Jul 10 17:34:41 1992
```

y^2 is a cubic having three real roots, say $\alpha_1<\alpha_2<\alpha_3$. We also assume, without loss of generality, that y^2 is monic. To reduce I_0, etc. to the three canonical integrals of Legendre, i.e., $F(\phi,k)$, $E(\phi,k)$, and $\Pi(\phi, \alpha^2,k)$, it is needed first the following transformations (Ref. 6)

$$x = \frac{a \sin^2\phi+b}{c \sin^2\phi+d}, \qquad 0\leq\phi\leq\pi/2$$

$$dx = \frac{2D \sin\phi\cos\phi}{(c \sin^2\phi+d)^2} d\phi, \tag{12}$$

$$y = G\frac{\sin\phi\cos\phi \ \Delta}{(c \sin^2\phi+d)^2}, \quad \Delta^2 = 1-k^2 \sin^2\phi$$

Here the values of a, b, c, d, k, D=ad-bc, and G are given parameters determined by the integration intervals. For example, for the integration interval $x>\alpha_3$, we have

$$a = -\alpha_2, \ b = \alpha_3, \ c = -1, \ d = 1,$$
$$D = \alpha_3,-\alpha_2, \ k^2 = (\alpha_2-\alpha_1)/(\alpha_3-\alpha_1),$$
$$G = (\alpha_3-\alpha_2)\sqrt{\alpha_3-\alpha_1}$$

We then may express I_0, I_1, and H_1 in terms of Legendre's normal integrals as follows

$$I_0(x) = \frac{2D}{G}F(\phi,k) + \text{const.} \tag{13}$$

$$I_1(x) = \frac{2D}{G}\{\alpha_3F(\phi,k)-(\alpha_3-\alpha_1)E(\phi,k) + (\alpha^3-\alpha_1)\Delta\tan\phi\}+ \text{const.} \tag{14}$$

$$H_1(x) = \frac{2D}{G(a-\gamma_1c)}\{cF(\phi,k) + \frac{D}{b-\gamma_1d}\Pi(\phi, \frac{a-\gamma_1c}{b-\gamma_1d})\} + \text{const.} \tag{15}$$

For the integration variable lying in the open or closed intervals (α_1,α_2), (α_2,α_3), and $(0,\alpha_1)$ these parameters will be different. In the case y^2 has one real and two complex roots, the transformation eq. (12) and the corresponding parameters will be entirely different.

Thus the case of y^2 being cubic may be summarized as follows. There are two sets of transformations depending on whether all roots are real. For each set it is necessary to distinguish four sub-intervals - a total of eight cases. The situation of y^2 being quartic requires three sets of transformations (4 real roots, 4 complex roots, and 2 real and 2 complex). All in all, it appears to us that Legendre's canonical reduction would require a fair amount of storage of auxilliary information. This makes it unattractive as a candidate for a mechanical reduction procedure. As mentioned in a previous report (Ref. 2), in Carlson's approach, these distinct cases are simplified through permutation symmetry.

The second alternative we consider is a reduction to Weirstrass' normal integrals. Again let us first discuss the simplest case, viz., $y^2 = a_3x^3+a_2x^2+a_1x+a_0$.

The substitution

$$x = (\frac{4}{a_3})^{1/3}t-\frac{a_2}{3a_3}$$

$$g_2 = (\frac{a_2^2}{3a_3} - a_1)(\frac{4}{a_3})^{1/3}$$

$$g_3 = \frac{a_1a_2}{3a_a} - \frac{2a_2^3}{27a_3^2} - a_0$$

would enable eq. (2) to be expressible in terms of Weirstrass' p-function $p(u)$, viz.,

$$I_0 = \int_\infty^{p(u)} \frac{dt}{(4t^3-g_2t-g_3)^{1/2}} \tag{16}$$

$$I_1 = \int_{p(u_0)}^{p(u)} \frac{t \ dt}{(4t^3-g_2t-g_3)^{1/2}} \tag{17}$$

$$H_1 = \int_\infty^{p(u)} \frac{dt}{(t-\gamma_1)(4t^3-g_2t-g_3)^{1/2}} \tag{18}$$

For y^2 being a quartic similar transformations may be applied to express I_0, I_1, and H_1 in similar terms. From practical considerations, we have deemed the Weirstrass' reduction procedure not appropriate as a candidate for symbolic integration, for two major reasons. First, the Weirstrass' p-functions or related functions are relatively unknown to most scientists and engineers. Second, the Weirstrass' functions arose as instances of doubly-periodic functions and as such were only related to the present problem as its inverse. As corrolary to this, there has not been much investigation on the computation of the inverse p-functions, and the unavailability of both numerical algorithms and software is considered a deterrent to the candidacy of Weirstrass' method for symbolic integration.

A third alternative in the canonical reduction involves the use of Jacobi's elliptic

Frontiers for Computer Algebra in AI

Richard Fateman
University of California at Berkeley

Abstract

The field of computer algebra has been spawned in part by application programs and in part by AI. The field has matured and grown substantially more algorithmic and system oriented, and as a consequence has for the most part has been diverted from its AI roots. It still seems worthwhile to consider the encoding of "mathematical knowledge" more generally than current systems; it also seems vital for the success of certain kinds of reasoning about physical systems to draw upon more powerful facilities for reasoning in mathematical analysis domains including intervals, constants, inequalities, singularities and complex variables, Several popular "modern" systems seem to have retreated (perhaps unintentionally) from these needs, and are less ambitious than earlier academic work.

1 Past and Current Work

The field of computer algebra has been spawned in part by application programs and in part by AI. Probably the first large Lisp program was James Slagle's 1961 doctoral work on symbolic indefinite integration. It approached the freshman–calculus domain as an AI search problem.

In the past 30 years this field has matured and grown substantially more algorithmic and oriented towards specific applications. Important new algorithms and representation techniques were developed, which, combined with far more powerful and low-cost computers, made computer algebra systems such as Axiom, Derive, Macsyma, Maple, Mathematica, Reduce widely distributed. On the down side, programmers have retreated from the goal of implementing "all of mathematics" symbolically and computationally. Yet some of the successes are still limited by the underlying ignorance of mathematics in the computer algebra systems When even slightly more general problem solving behavior is needed that the limited and somewhat ad hoc methods in today's systems fail. (List (cos x) (sin x)) (x 0 10)

2 What don't systems know?

It is usually recognized that one critical issue of symbolic mathematics is being able to determine if two expressions denote the same formula. A trivial example is to realize that x^2-1 and $(x-1)*(x+1)$ are the same function. Most systems can do that. Another trivial example is that $(x^2-1)/(x+1)$ and $(x-1)$ are the same functions *except possibly* at $x=-1$. Most systems, including those mentioned above fail at this task.

As another example, you may recall from your mathematics courses that $\log(xy) = \log(x) + \log(y)$, and may be disappointed by some computer algebra systems' apparent inability to apply this simplification. Systems that don't automatically make this change can be instructed to do so. Unfortunately, for $x = y = -1$, this alleged equation tells us that $0 = -2i\pi$.

Any of the ambitious computer systems can be provoked (some more easily than others) into committing numerous erroneous calculation involving incorrect choice of branch cuts, divisions by zero or infinity, failures to check for continuity of functions, etc.

There are consistent subsets in each system – for example one can usually trust exact integer and rational arithmetic, polynomial computations in one or several variables over a field, so all is not lost.

Most systems attempt to compute in domains that are less contrived and more appropriate for applications. But if you introduce only a modest set of additional functions – trignometric functions, square-root, and the constant π will do, then it has been shown that the equivalence-to-zero predicate is undecidable. Far short of such complications, however, most contemporary systems can be provoked into errors.

3 Why do we care?

What is the relevance of this? Any AI or scientific computation system dealing with automatic symbolic mathematics ranging beyond the most straightforward polynomial arithmetic can run into problems. If algebraic manipulations are made in ignorance of domains (e.g. manipulations in complex plane), all existing computer algebra systems can be guided into outrageous blunders. Correcting these problems requires careful thought and re-design. There is also some synergy with theorem proving, since one way of determining that a simplification is correct is to – in effect– prove that it is correct in every domain.

Current work at Berkeley involves representation and computation problems facing systems that must manipulate functions like log, or square-root, with branch-cuts in the complex domain. Without solutions for such problems, it appears hopeless to expect advanced algorithms such as the Risch integration procedure to be useful as black-box utilities in real-life GENERAL applications. Reasoning about real intervals, compositions of functions in the complex plane, representations of finite and infinite geometric domains, and limits of continuous functions all seem to be needed.

If the domains are restricted by the problem solving domain (e.g. in the example above, if x and y are necessarily real, or have positive real part, the log simplification works), we have what could, somewhat facetiously, be called ignorance-based tools. That is, if the user is ignorant of the meaning of log in the complex plane, and replaces $\log x$ by $\log |x|$ – an interpretation sometimes used in freshman calculus– then perhaps the system can be ignorant of such interpretations as well. Unfortunately, ignorance of complex numbers is quite risky since they are easily introduced by square-roots of negative numbers.

While I believe that there are excellent opportunities for linking applications of symbolic mathematical manipulation to the development of models, programs, and data analysis in large-scale computing (including parallel systems) these seem to be mostly in the talking stage. Many applications programs are designed so that the principal realizable direction for progress is well-known: a computer that is faster will push the frontier.

I look forward to seeing any systems for symbolic mathematics or AI that have proceeded to the point of pushing an applied computing frontier forward in a direction that is either different from the conventional one, or pushing it much further than could be expected by merely providing faster numerical computing. At one time it was thought that magnetic fusion energy computing requirements would push us into smarter rather than faster computing; it may still be true that this will happen. Perhaps program verification will be a winner. Perhaps some kind of computer-aided design or programming in some applied field will win. If this symposium smokes out such situations that would be really quite exciting.

How can AI help solve research problems? New algorithmic ideas/ relevance to conventional AI Integration How can numerical methods provide info near symbolic results Code generation?

peoplesparc.Berkeley.EDU:fateman

stdin

Fri Jul 10 17:35:20 1992

cx / cx lw507 Evans

```
cx peoplesparc.Berkeley.EDU:fateman  Job: stdin  Date: Fri Jul 10 17:35:20 1992

cx peoplesparc.Berkeley.EDU:fateman  Job: stdin  Date: Fri Jul 10 17:35:20 1992

cx peoplesparc.Berkeley.EDU:fateman  Job: stdin  Date: Fri Jul 10 17:35:20 1992

cx peoplesparc.Berkeley.EDU:fateman  Job: stdin  Date: Fri Jul 10 17:35:20 1992
```